CSE211 Digital Design

Akdeniz University

Week10: Combinational Logic Part 1

Assoc.Prof.Dr. Taner Danışman

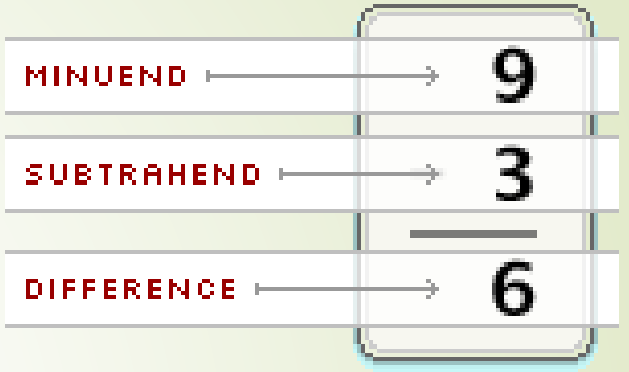
tdanisman@akdeniz.edu.tr

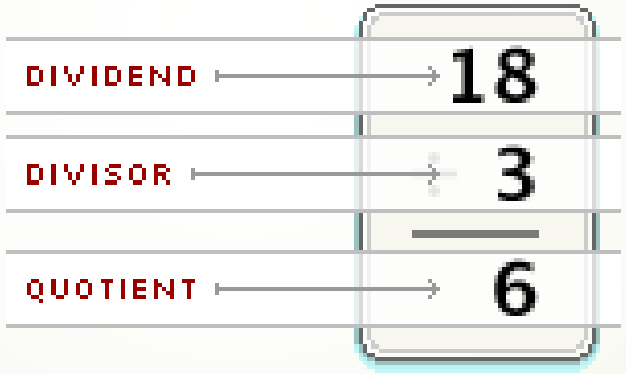
Course program

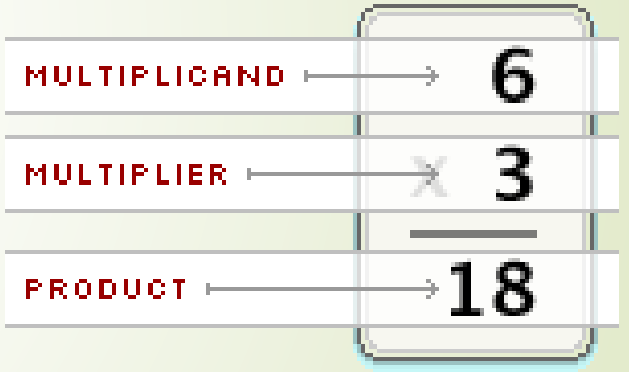
Week 01	2-Oct-23	Introduction
Week 02	9-Oct-23	Digital Systems and Binary Numbers I
Week 03	16-Oct-23	Digital Systems and Binary Numbers II
Week 04	23-Oct-23	Boolean Algebra and Logic Gates I
Week 05	30-Oct-23	Boolean Algebra and Logic Gates II
Week 06	6-Nov-23	Gate Level Minimization
Week 07	13-Nov-23	Karnaugh Maps
Week 08	20-Nov-23	Midterm
Week 09	27-Nov-23	Karnaugh Maps
Week 10	4-Dec-23	Combinational Logic
Week 11	11-Dec-23	Combinational Logic
Week 12	18-Dec-23	Timing, delays and hazards
Week 13	25-Dec-23	Synchronous Sequential Logic
Week 14	1-Jan-24	Synchronous Sequential Logic

Terms to know

- ASIC
- MSI
- VLSI



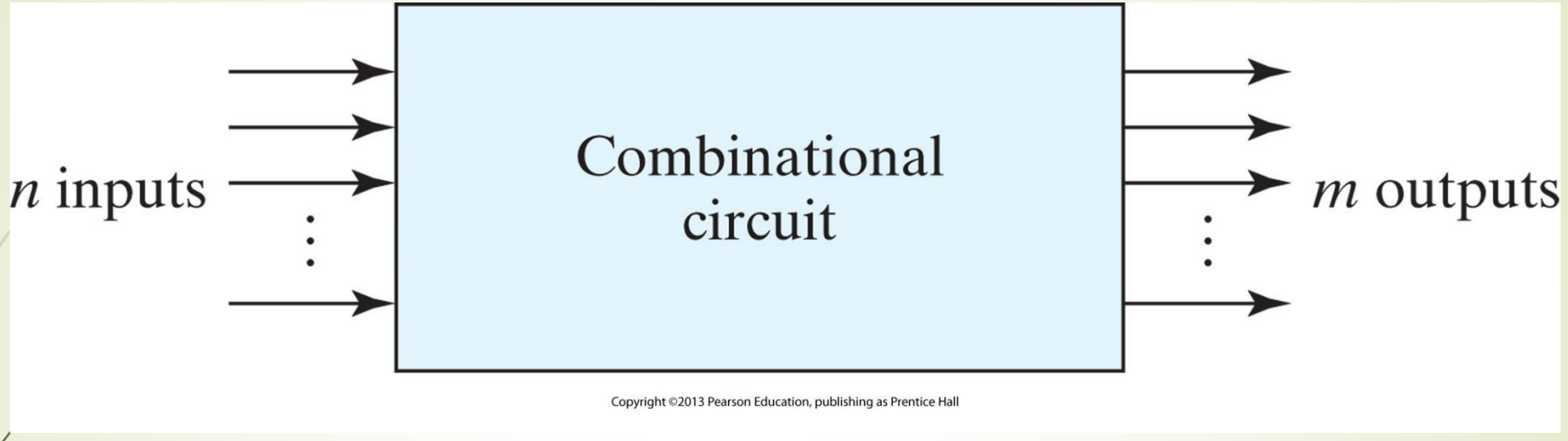




Classifications

- **Combinational Logic**
  - **No memory** unit
  - Expressed by Boolean functions
  - Outputs depends on only the **present** inputs
- **Sequential Logic**
  - Storage elements + logic gates
  - The content of the storage elements define the state of the circuit
  - Outputs are functions of both input and current state
  - State is a function of previous inputs
  - Outputs not only depend the present inputs but also the past inputs
  - Time sequence of inputs and internal states

Combinational Circuit



- **n** input bits ➔ **2^n** possible binary input combinations
- **m** output variable produced
  - For each possible input combination, there is one possible output value
    - truth table
    - Boolean functions (with n input variables)
- **Examples: adders, subtractors, comparators, decoders, encoders, and multiplexers.**

So Far...

- **Chapter #1** binary numbers and binary codes that represent discrete quantities of information.
- **Chapter #2** Boolean algebra as a way to express logic functions algebraically
- **Chapter #3** How to simplify Boolean functions to achieve economical (simpler) gate implementations.
- **Chapter #4** to use the knowledge acquired in previous chapters to formulate systematic analysis and design procedures for combinational circuits
  - Analyze the behavior of a given logic circuit
  - synthesize a circuit that will have a given behavior
  - Write hardware description language (HDL) models

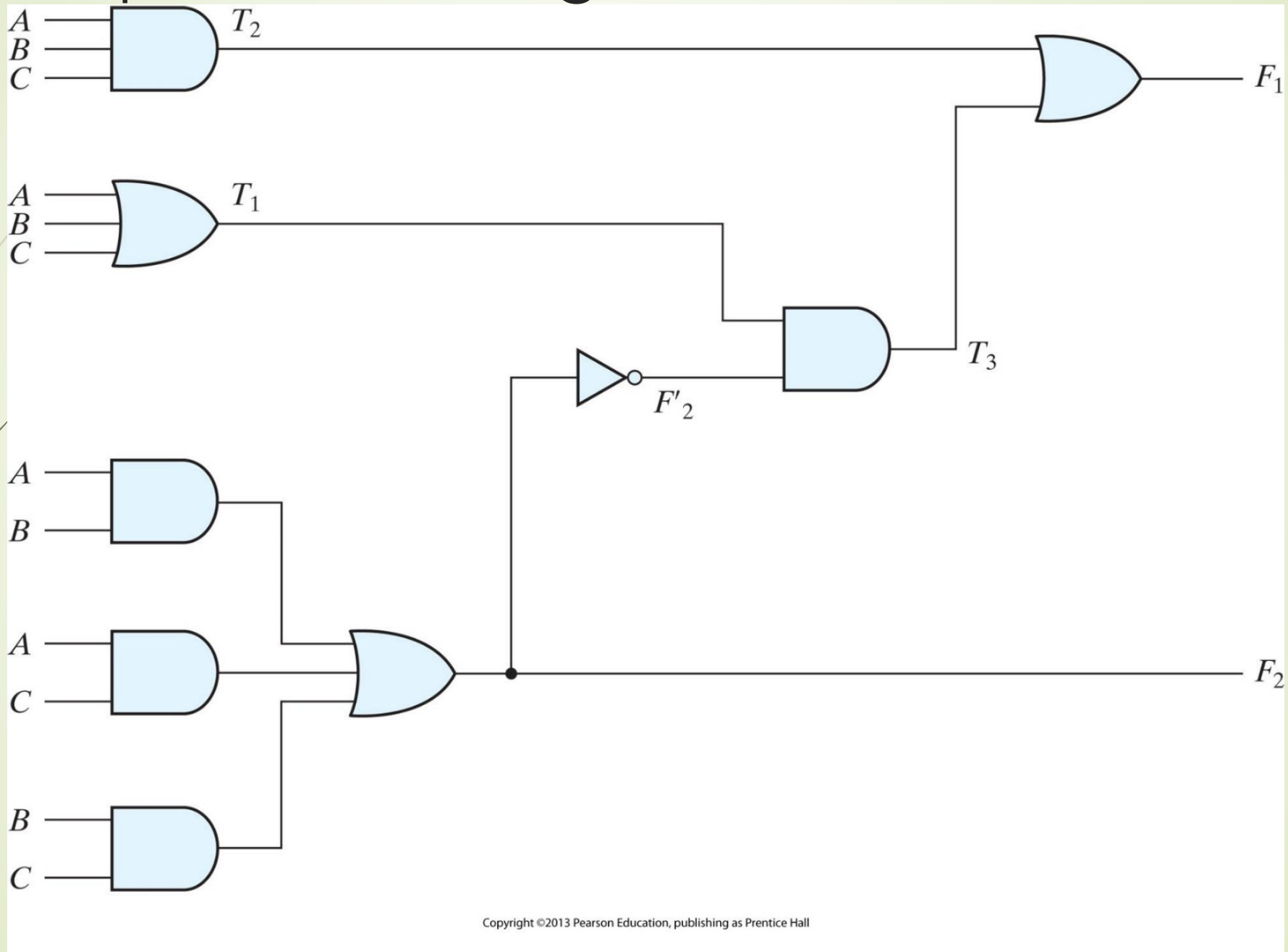
Analysis & Design of Combinational Logic

- **<u>Analysis</u>: to find out the function that a given circuit implements**
  - We are given a logic circuit and we are expected to find out
    1. Boolean function(s)
    2. Truth table
    3. A possible explanation of the circuit operation (i.e. what it does)
- Firstly, make sure that the given circuit is, **indeed**, **combinational**.

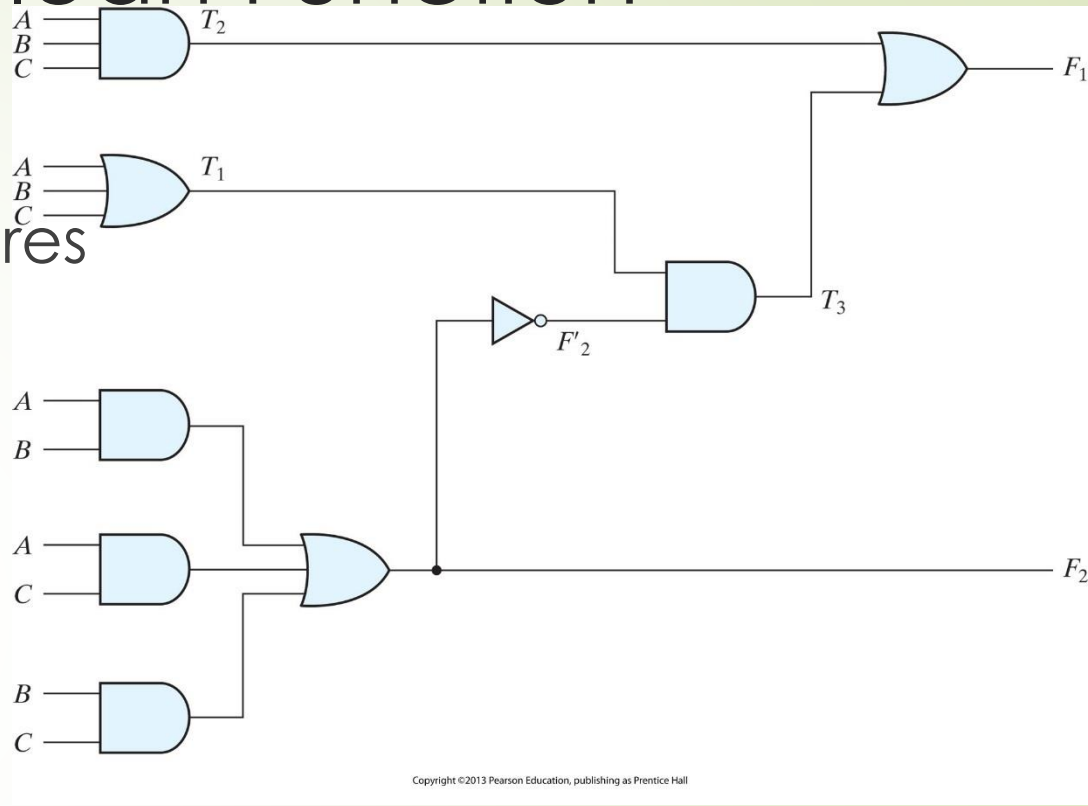
Analysis of Combinational Logic

- Verifying the circuit is combinational
  - Diagram has **no memory elements**
  - **No feedback paths** (connections)
    - A feedback path is a connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate
- Secondly, obtain a Boolean function for each output or the truth table
- Lastly, interpret the operation of the circuit from the derived Boolean functions or truth table
  - What is it the circuit doing?
  - Addition, subtraction, multiplication, etc.
- The analysis can be performed by using a computer program

Example: Obtaining Boolean Function



Example: Obtaining Boolean Function



- Boolean expressions for named wires
  - $T_1 = a + b + c$
  - $T_2 = abc$
  - $F_2 = ab + ac + bc$
  - $T_3 = F'_2 T_1 = (ab + ac + bc)' (a + b + c)$
  - $F_1 = T_2 + T_3$

    $= abc + (ab + ac + bc)' (a + b + c)$

    $= abc + ((a' + b')(a' + c')(b' + c')) (a + b + c)$

    $= abc + ((a' + a'c' + a'b' + b'c')(b' + c')) (a + b + c)$

    $= abc + (a'b' + a'c' + a'b'c' + b'c') (a + b + c)$

Example: Obtaining Boolean Function

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

- This is what we call **full-adder** (FA)

Design of Combinational Logic

- We start with the **verbal specification** about what the resulting circuit will do for us (i.e. which function it will implement)
  - Specifications are often verbal, and **very likely incomplete** and **ambiguous** (**if not faulty**)
  - **Wrong interpretations** can result in **incorrect circuit**
- We are expected to find
  1. **Boolean function(s)** (or **truth table**) to realize the desired functionality
  2. **Logic circuit** implementing the Boolean function(s) (or the truth table)

Design Constraints

- From the truth table, we can obtain a variety of simplified expressions
- Question: which one to choose?
- The design constraints may help in the selection process
- **Constraints:**
  - number of gates
  - propagation time of the signal all the way from the inputs to the outputs
  - number of inputs to a gate
  - number of interconnections
  - power consumption
  - driving capability of each gate

Code Conversion Example

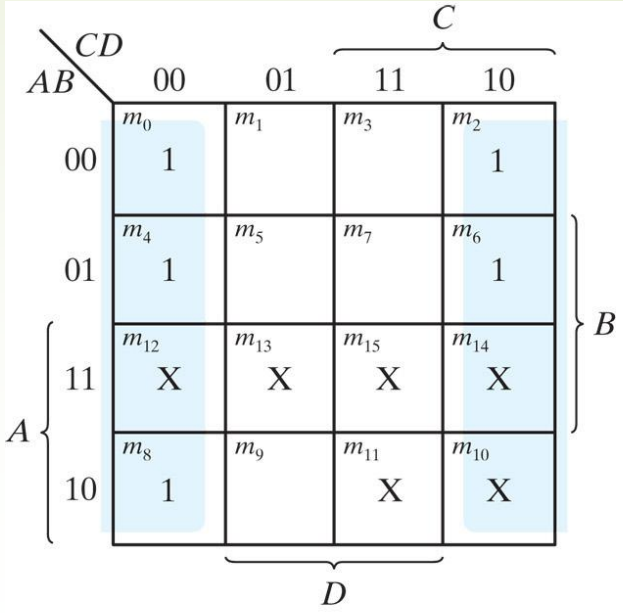
- To convert from binary code A to binary code B
- A combinational circuit performs this transformation by means of logic gates

Table 4.2
Truth Table for Code Conversion Example

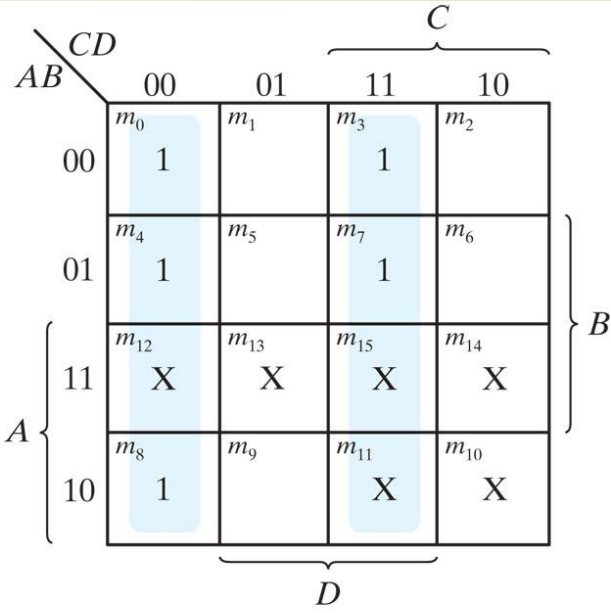
Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Code Conversion Example

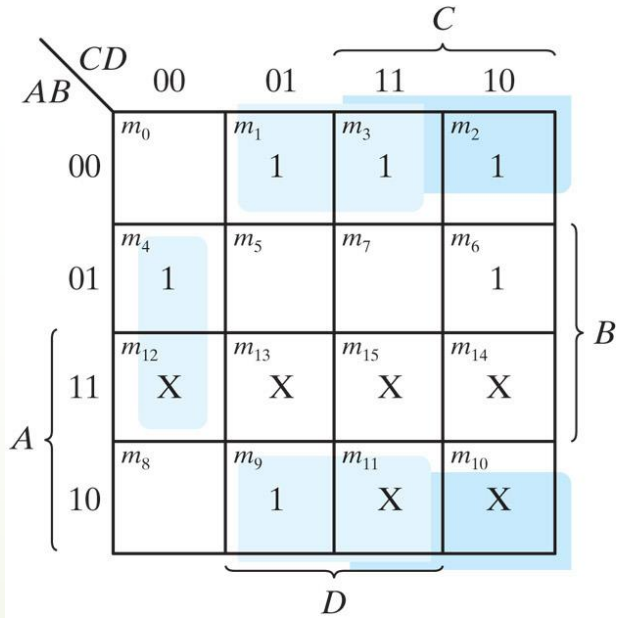
- **Find the boolean functions for w, x, y, and z**



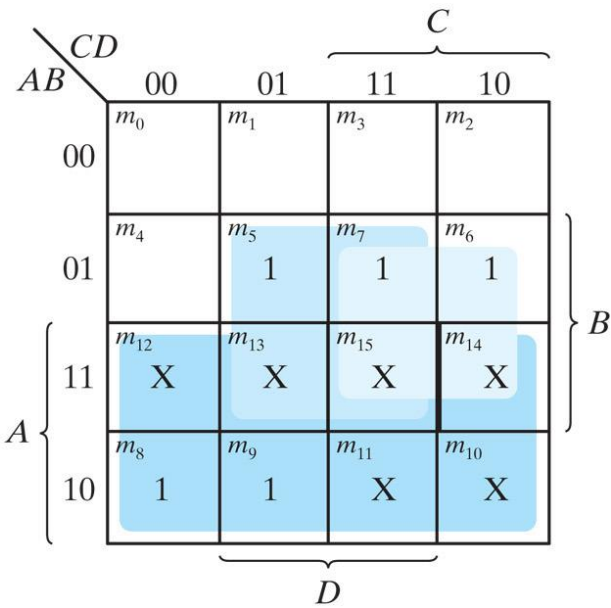
$z = D'$



$y = CD + C'D'$

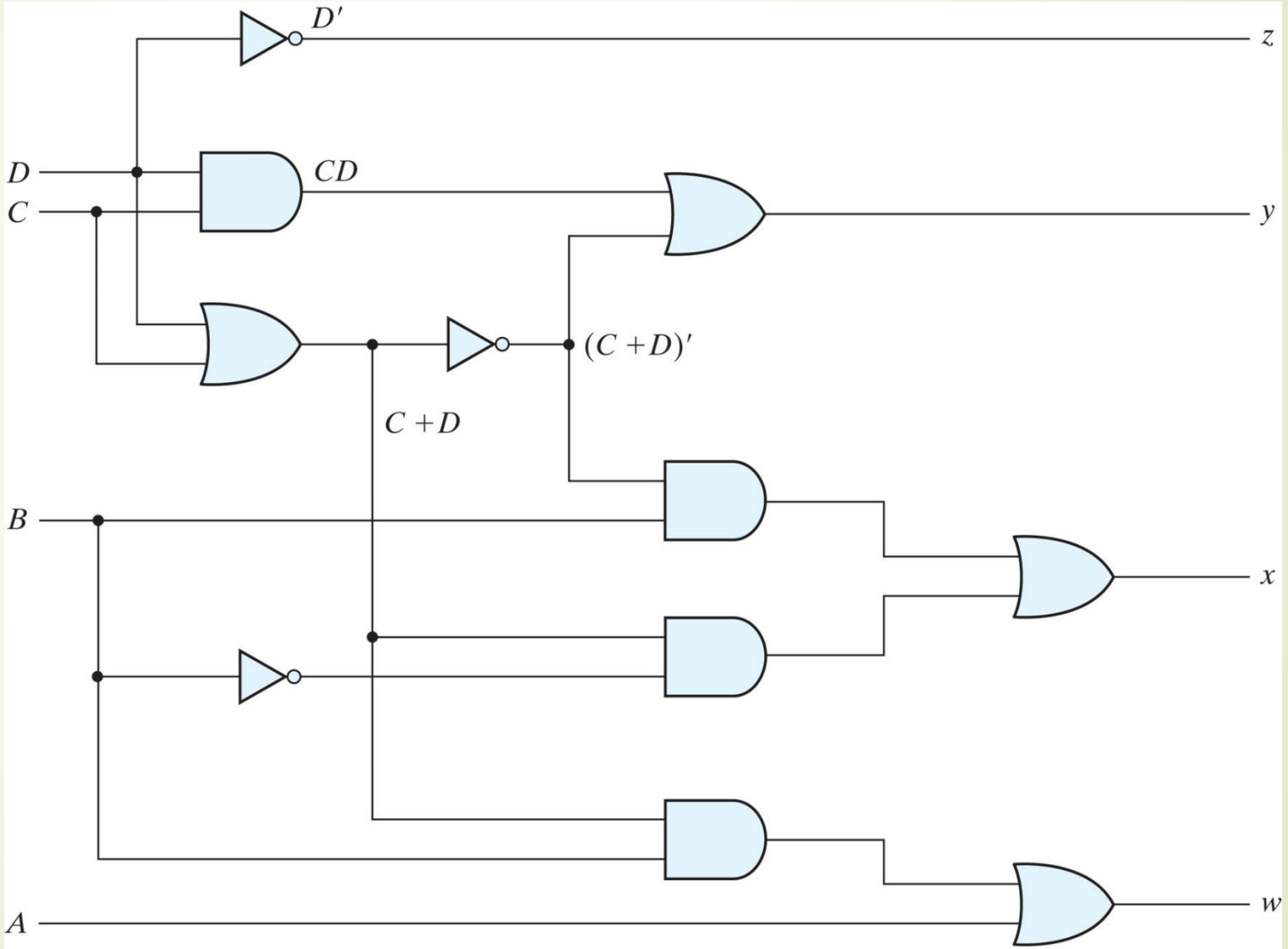


$x = B'C + B'D + BC'D'$



$w = A + BC + BD$

Logic diagram for BCD-to-excess-3 code converter



Binary Adder and Subtractor

- **(Arithmetic) Addition** of two binary digits
  - 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10
  - The result has two components
    - the sum (S)
    - the carry (C)
- A combinational circuit that performs the addition of two bits is called a ***half adder***
- One that performs the addition of three bits (two significant bits and a previous carry) is a ***full adder***
- **Two half adders** can be employed to implement a **full adder**.

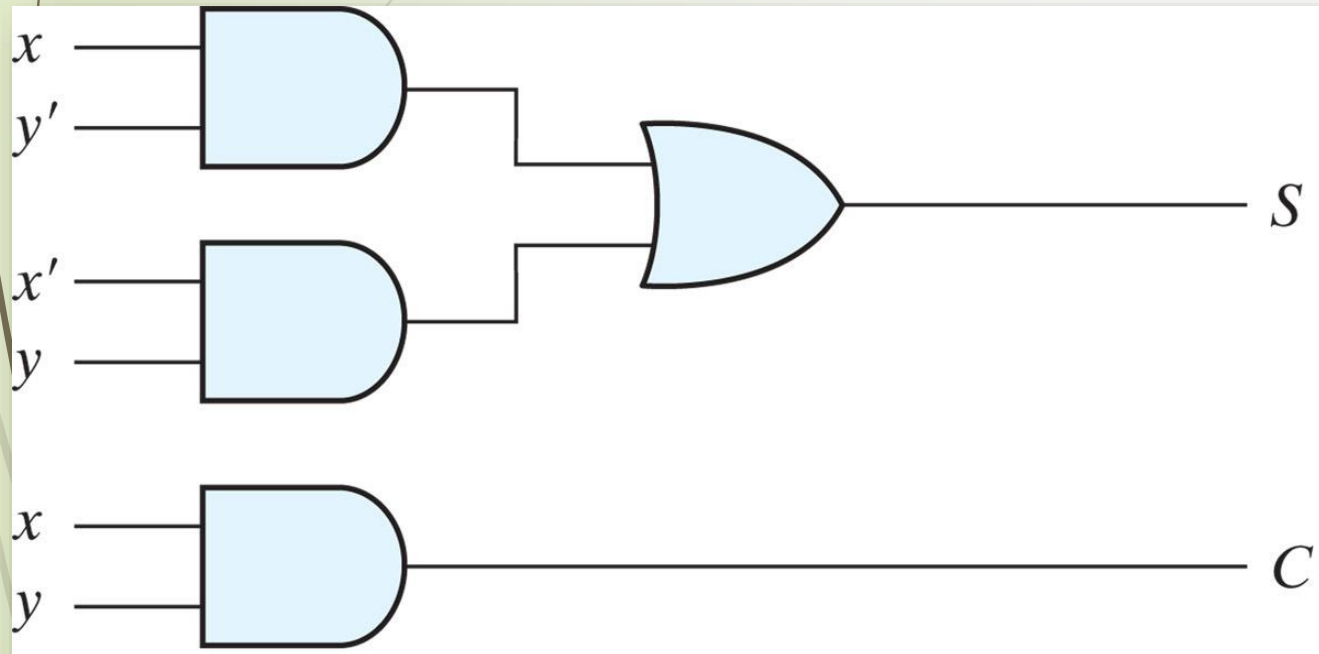
Half Adder

- The result has two components
  - the sum (S)
  - the carry (C)

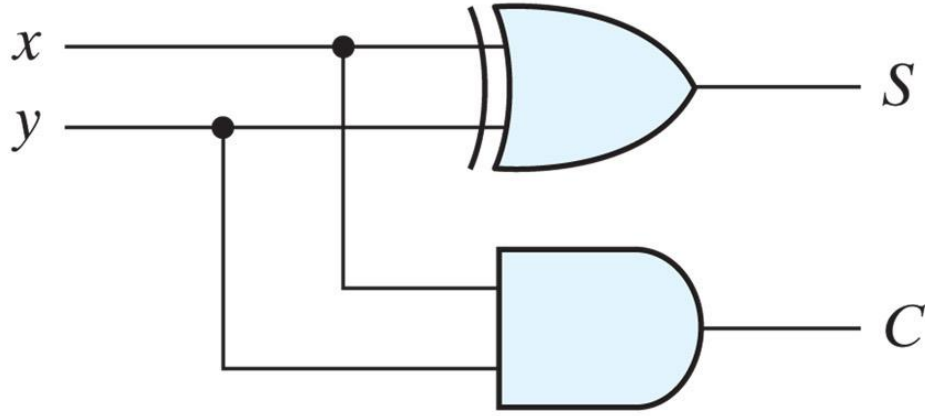
Table 4.3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder (Cont.)



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

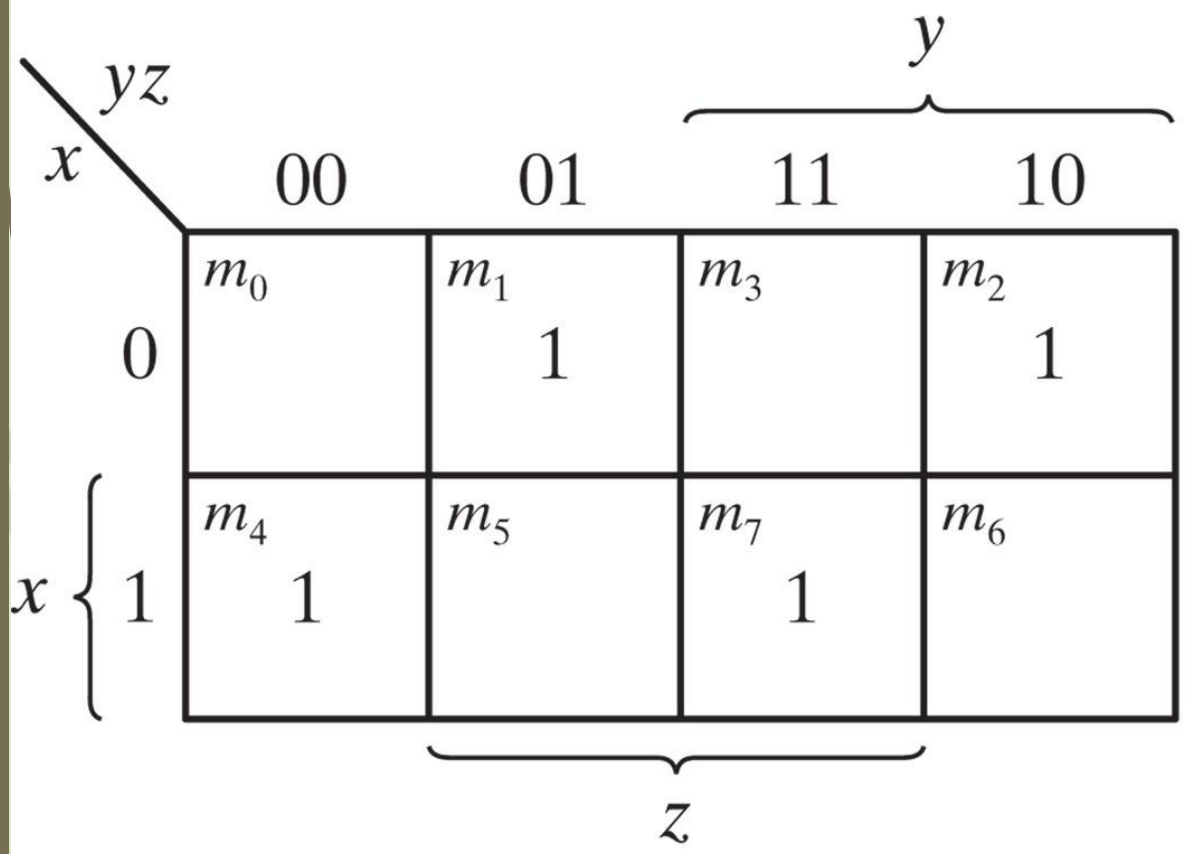
Full Adder

- A circuit that performs the arithmetic sum of **three bits**
  - Three inputs
  - the range of output is **[0, 3]**
  - Two binary outputs

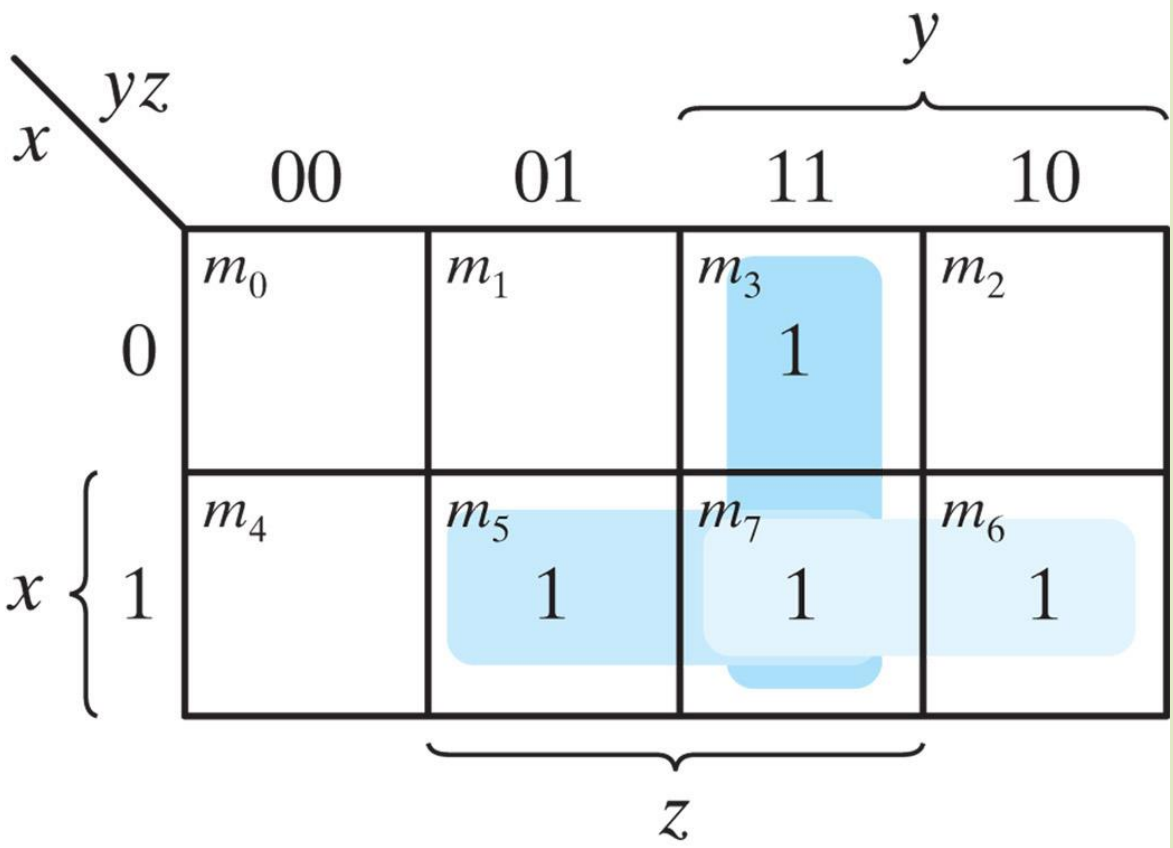
Table 4.4
Full Adder

x	*y*	*z*	*C*	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Map for Full Adder

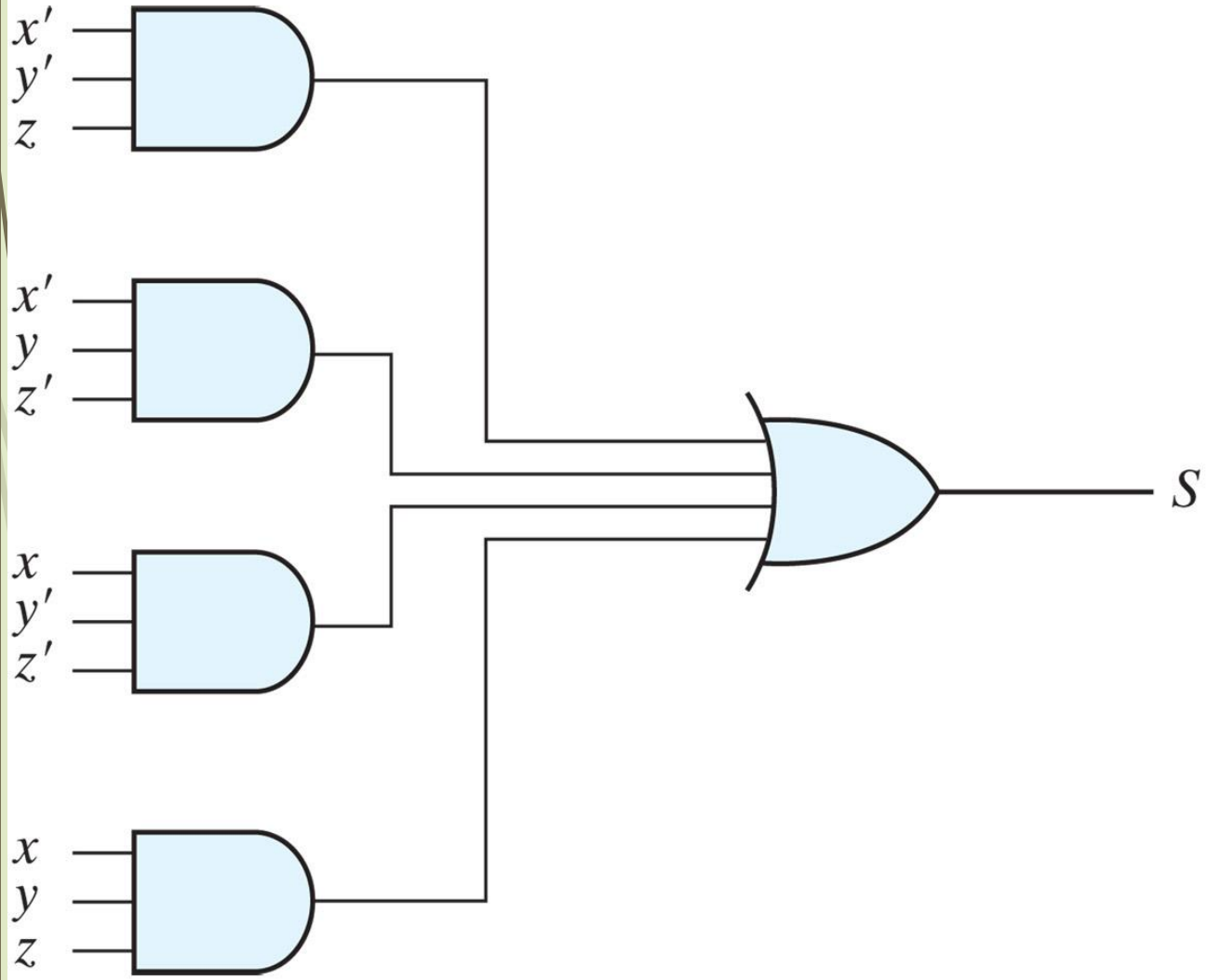


(a) $S = x'y'z + x'yz' + xy'z' + xyz$

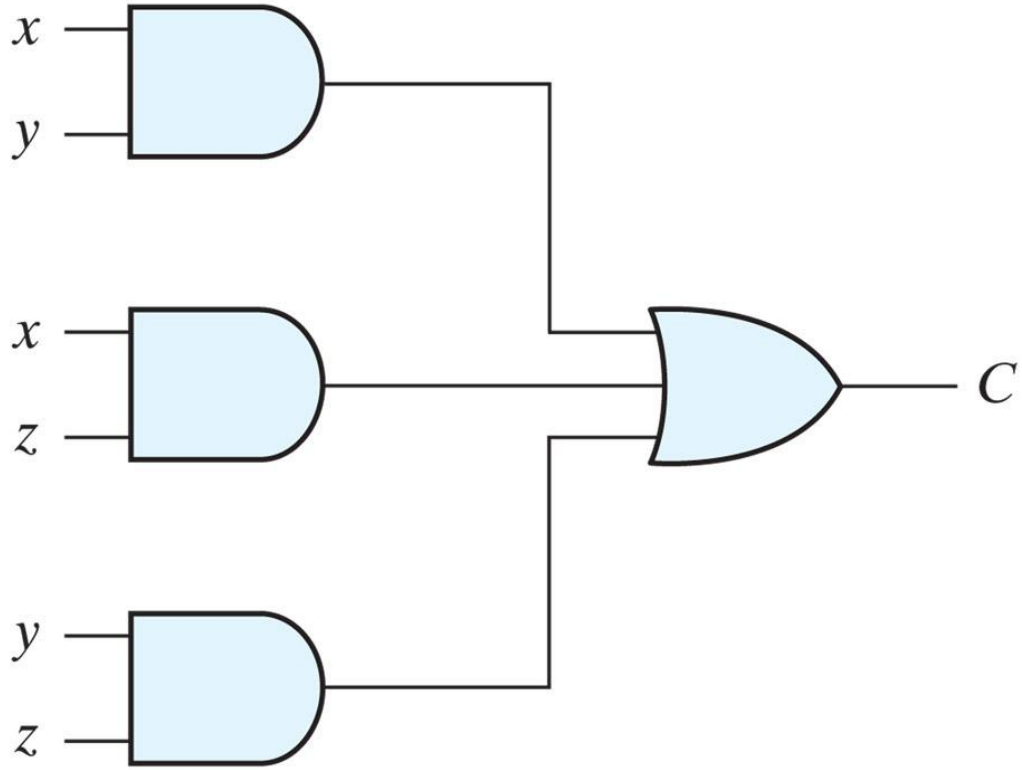


(b) $C = xy + xz + yz$

Implementation of full adder in sum-of-products form

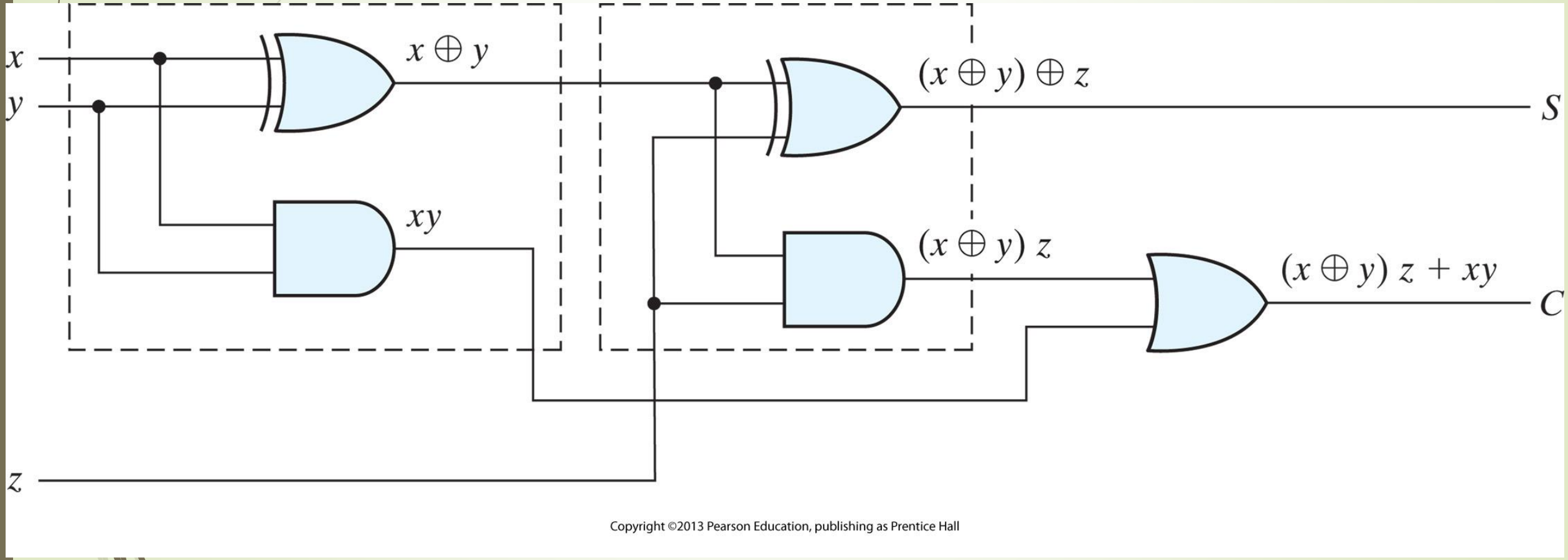


(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

Implementation of full adder with two half adders and an OR gate



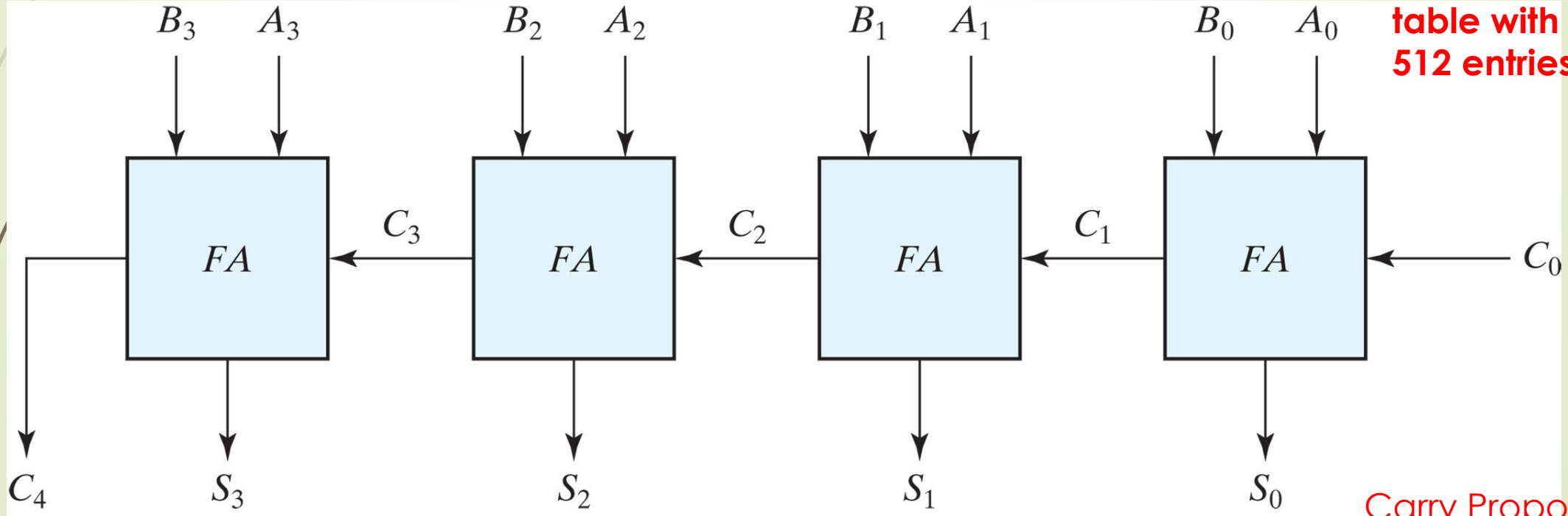
Binary Adders

- The design methodology we used to build carry-ripple adder is what is referred as **hierarchical design.**
- To add two 4-bit numbers, in classical design, we have:
  - 9 inputs including C_0.
  - 5 outputs
  - Truth tables with $2^9 = 512$ entries
  - We have to optimize five Boolean functions with 9 variables each.
- **In Hierarchical design**
  - we divide our design into smaller functional blocks
  - connect functional units to produce the big functionality

Binary adders

- A digital circuit that produces the arithmetic sum of two binary numbers
  - A = (a_{n-1}, a_{n-2}, …, a_1, a_0)
  - B = (b_{n-1}, b_{n-2}, …, b_1, b_0)
- A simple case: 4-bit binary adder

Design of this circuit by the classical method would require a truth table with 2^9 = 512 entries



Carry Propogation!

Binary Adders

- Consider the two binary numbers ***A* = 1011**, and ***B* = 0011**. Their sum ***S* = 1110** is formed with the four-bit adder as follows

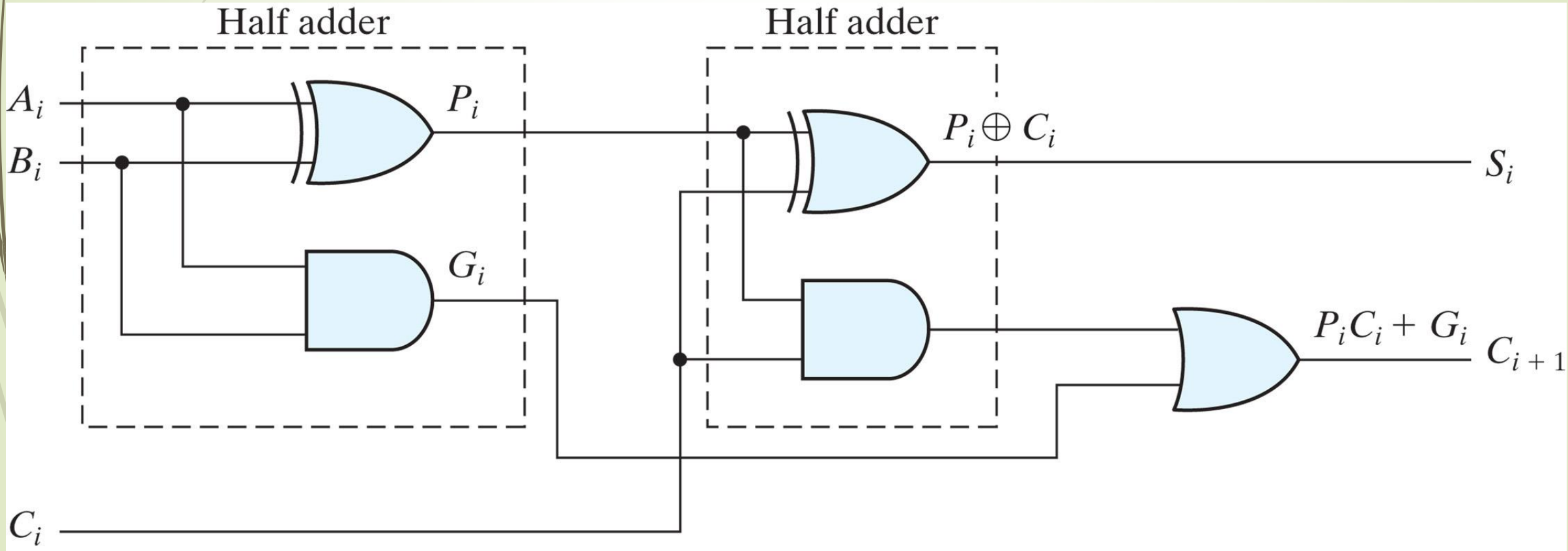
Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Carry Propogation

- The addition of two binary numbers in parallel implies that all the bits of the **augend** and **addend** are available for computation at the same time
- Sum output will be in its **steady-state final value** only after the input carry to that stage has been propagated
- The **outputs will not be correct** unless the signals are given enough time to **propagate** through the gates connected from the inputs to the outputs
- Solution is to use **carry lookahead logic**

Carry Propogation

- G_i is called a ***carry generate***
- P_i is called a ***carry propagate***



Carry Lookahead Generator

- C_3 does not have to wait for C_2 and C_1 to propagate
- This **gain** in speed of operation is achieved at the expense of additional **complexity** (hardware).

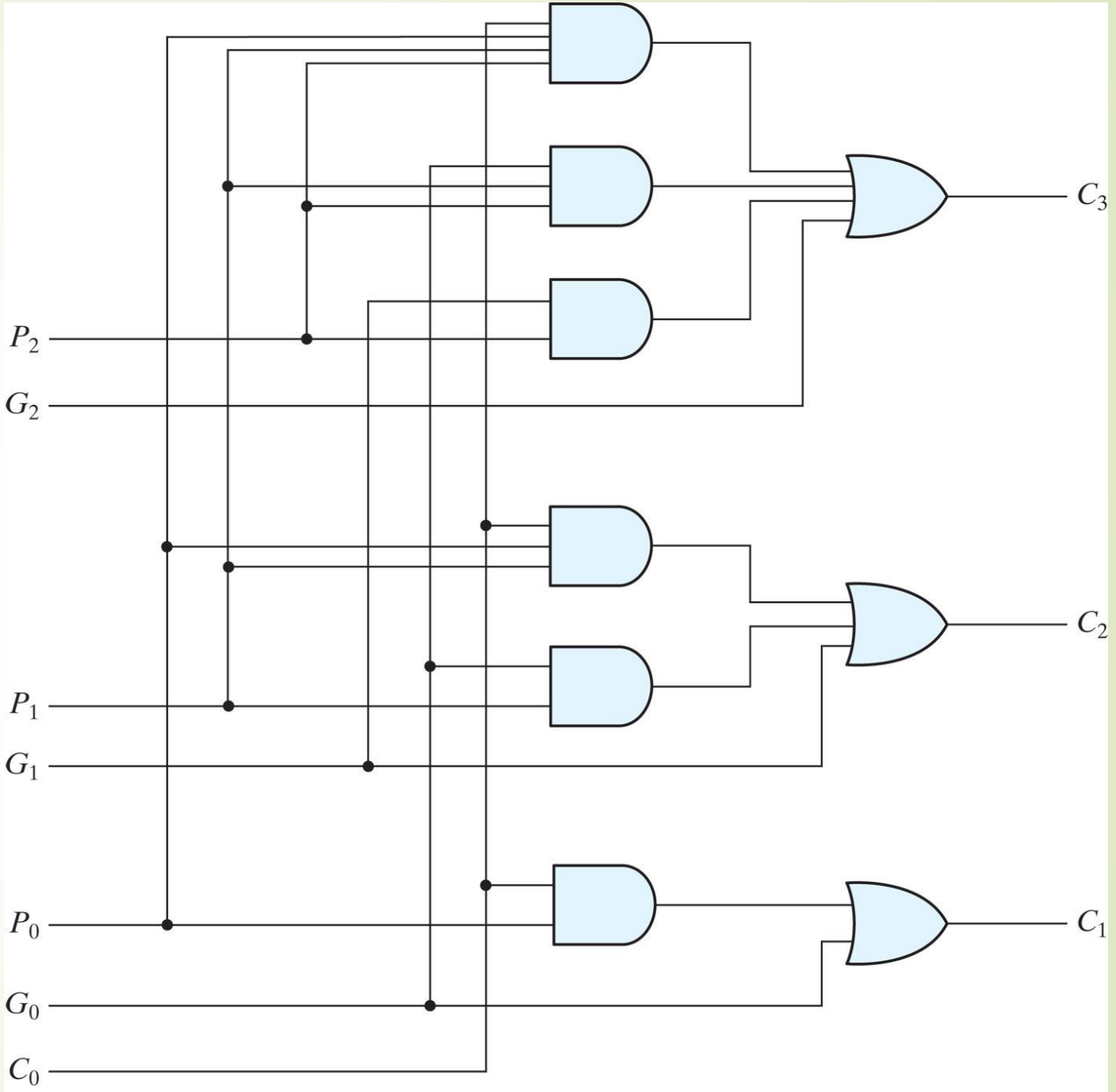
$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

Logic diagram of carry lookahead generator



Four-bit adder with carry look ahead